Python: module cdms.hgrid

cdms.hgrid

index

CDMS HorizontalGrid objects

Modules

MA Numeric PropertiedClasses cdms.bindex

Classes

cdms.grid.AbstractGrid(cdms.cdmsobj.CdmsObj)
AbstractHorizontalGrid
AbstractCurveGrid
DatasetCurveGrid
FileCurveGrid
TransientCurveGrid

class ***AbstractCurveGrid***(AbstractHorizontalGrid)

Method resolution order:
AbstractCurveGrid
AbstractHorizontalGrid
cdms.grid.AbstractGrid
cdms.cdmsobj.CdmsObj
cdms.internattr.InternalAttributesClass
PropertiedClasses.Properties.PropertiedClass

Methods defined here:

__init__(self, latAxis, lonAxis, id=None, maskvar=None, tempmask=None, node=None)
Create a curvilinear grid.

__repr__(self)

__str__ = __repr__(self)

checkAxes(self, axes)
Return 1 iff every element of getAxisList() is in the list 'a

clone(self, copyData=1)

flatAxes(self)

```
    Return (flatlat, flatlon) where flatlat is a 1D NumPy array
    having the same length as the number of cells in the grid, si
    for flatlon.
```

genBounds(self)
```
    # Don't try to generate bounds for curvilinear grids
```

getAxis(self, naxis)
```
    # Get the n-th index axis. naxis is 0 or 1.
```

getAxisList(self)

getGridSlices(self, domainlist, newaxislist, slicelist)
```
    Determine which slices in slicelist correspond to the lat/lon
    of the grid.
    domainlist is a list of axes of a variable.
    newaxislist is a list of result axes after the slicelist is a
    slicelist is a list of slices.

    All lists are of equal length.

    Return value is (newslicelist, gridaxislist) where
    newslicelist is the elements of slicelist that correspond to
      preferred order of the grid.
    gridaxislist is the elements of newaxislist that correspond t
      preferred order of the grid.
```

getIndex(self)
```
    Get the grid index
```

getMask(self)
```
    Get the mask array, if any, otherwise None is returned.
```

getMesh(self, transpose=None)
```
    Generate a mesh array for the meshfill graphics method.
    If transpose is defined to a tuple, say (1,0), first transpos
    latbounds and lonbounds according to the tuple, (1,0,2) in th
```

intersect(self, spec)
```
    Intersect with the region specification.

    'spec' is a region specification of the form defined in the g

    Returns (mask, indexspecs) where
    'mask' is the mask of the result grid AFTER self and region s
    'indexspecs' is a list of index specifications suitable for s
      variable with the given grid.
```

isClose(self, g)
```
    Return 1 iff g is a grid of the same type and shape. A real e
    comparison would be too expensive here.
```

reconcile(self, axes)

```
Return a grid that is consistent with the axes, or None.
For curvilinear grids this means that the grid-related axes a
contained in the 'axes' list.
```

size(self)

subSlice(self, *specs, **keys)

```
Get a transient subgrid based on an argument list <specs> of
```

toCurveGrid(self, gridid=None)

toGenericGrid(self, gridid=None)

writeScrip(self, cufile, gridTitle=None)

```
Write a grid to a SCRIP file.
cufile is a Cdunif file, NOT a CDMS file.
gridtitle is a string identifying the grid.
```

writeToFile(self, file)

Methods inherited from AbstractHorizontalGrid:

checkConvex(self)

```
Check that each cell of the grid is convex in lon-lat space,
Return a 1D Numeric array of cells that fail the cross-produc
```

fixCutCells(self, nonConvexCells, threshold=270.0)

```
For any mapping from a spherical to a planar surface, there i
Grid cells that span the cut may appear to be nonconvex, whic
problems with meshfill graphics. This routine attempts to 're
boundaries so that meshfill recognizes they are convex.

nonConvexCells: 1D Numeric array of indices of nonconvex cell
  checkConvex.
threshold: positive floating-point value in degrees.
  If the difference in longitude values of
  consecutive boundaries nodes exceeds the threshold, the cel
  a cut cell.

On return, the grid boundaries are modified.
Return value is a 1D array of indices of cells that cannot be
```

getBounds(self)

```
Get the grid cell boundaries, as a tuple (latitudeBounds, lon
```

getLatitude(self)

```
Get the latitude coordinates.
```

getLongitude(self)

```
Get the longitude coordinates.
```

getWeightsArray(self)

```
Return normalized area weights, as an array of the same
shape as the grid.
```

hasCoordType(self, coordType)

listall(self, all=None)

setMask(self, mask, permanent=0)

subGridRegion(self, latRegion, lonRegion)

Methods inherited from cdms.grid.AbstractGrid:

info(self, flag=None, device=None)

```
Write info about slab; include dimension values and weights i
```

Methods inherited from cdms.cdmsobj.CdmsObj:

dump(self, path=None, format=1)

```
dump(self,path=None,format=1)
Dump an XML representation of this object to a file.
'path' is the result file name, None for standard output.
'format'==1 iff the file is formatted with newlines for reada
```

matchPattern(self, pattern, attribute, tag)

```
# Match a pattern in a string-valued attribute. If attribute
# search all string attributes. If tag is not None, it must m
```

matchone(self, pattern, attname)

```
# Return true iff the attribute with name attname is a string
# attribute which matches the compiled regular expression pat
# if attname is None and pattern matches at least one string
# attribute. Return false if the attribute is not found or is
```

searchPattern(self, pattern, attribute, tag)

```
# Search for a pattern in a string-valued attribute. If attri
# search all string attributes. If tag is not None, it must m
```

searchPredicate(self, predicate, tag)

```
# Apply a truth-valued predicate. Return a list containing a
# if the predicate is true and either tag is None or matches
# If the predicate returns false, return an empty list
```

searchone(self, pattern, attname)

```
Return true iff the attribute with name attname is a string
attribute which contains the compiled regular expression patt
if attname is None and pattern matches at least one string
attribute. Return false if the attribute is not found or is n
a string.
```

Methods inherited from cdms.internattr.InternalAttributesClass:

is_internal_attribute(self, name)
```
    is_internal_attribute(name) is true if name is internal.
```

replace_external_attributes(self, newAttributes)
```
    replace_external_attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes
```

Methods inherited from PropertiedClasses.Properties.PropertiedClass:

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)
```
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.
```

get_property_g(self, name)
```
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.
```

get_property_s(self, name)
```
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.
```

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
```
    Set attribute handlers for  name to methods actg, acts, actd
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.
        nowrite defaults to 0.
    nodelete = 1 prevents deleting this attribute.
        nodelete defaults to 1 unless actd given.
    if nowrite and nodelete is None: nodelete = 1
```

class ***AbstractHorizontalGrid***(cdms.grid.AbstractGrid)

Method resolution order:
- AbstractHorizontalGrid
- cdms.grid.AbstractGrid
- cdms.cdmsobj.CdmsObj
- cdms.internattr.InternalAttributesClass
- PropertiedClasses.Properties.PropertiedClass

Methods defined here:

__init__(self, latAxis, lonAxis, id=None, maskvar=None, tempmask=None, node=None)

```
    Create a horizontal grid.
```

checkConvex(self)
```
    Check that each cell of the grid is convex in lon-lat space,
    Return a 1D Numeric array of cells that fail the cross-produc
```

fixCutCells(self, nonConvexCells, threshold=270.0)
```
    For any mapping from a spherical to a planar surface, there i
    Grid cells that span the cut may appear to be nonconvex, whic
    problems with meshfill graphics. This routine attempts to 're
    boundaries so that meshfill recognizes they are convex.

    nonConvexCells: 1D Numeric array of indices of nonconvex cell
      checkConvex.
    threshold: positive floating-point value in degrees.
      If the difference in longitude values of
      consecutive boundaries nodes exceeds the threshold, the cel
      a cut cell.

    On return, the grid boundaries are modified.
    Return value is a 1D array of indices of cells that cannot be
```

genBounds(self)
```
    # Generate default bounds
```

getAxis(self, naxis)
```
    # Get the n-th axis. naxis is 0 or 1.
```

getBounds(self)
```
    Get the grid cell boundaries, as a tuple (latitudeBounds, lon
```

getLatitude(self)
```
    Get the latitude coordinates.
```

getLongitude(self)
```
    Get the longitude coordinates.
```

getMask(self)
```
    Get the mask array, if any, otherwise None is returned.
```

getMesh(self)
```
    Get the mesh array used by the meshfill plot.
```

getWeightsArray(self)
```
    Return normalized area weights, as an array of the same
    shape as the grid.
```

hasCoordType(self, coordType)

listall(self, all=None)

setMask(self, mask, permanent=0)

subGridRegion(self, latRegion, lonRegion)

Methods inherited from cdms.grid.AbstractGrid:

__repr__ = __str__(self)

__str__(self)

checkAxes(self, axes)
```
Return 1 iff self.getAxisList and axes are consistent.
```

clone(self, copyData=1)
```
Make a copy of self.
```

flatAxes(self)
```
Return (flatlat, flatlon) where flatlat is a raveled NumPy ar
having the same length as the number of cells in the grid, si
for flatlon.
```

getAxisList(self)

info(self, flag=None, device=None)
```
Write info about slab; include dimension values and weights i
```

isClose(self, g)
```
Return 1 if g is 'close enough' to self to be considered equa
```

reconcile(self, axes)
```
Return a grid that is consistent with the axes, or None.
```

size(self)
```
Return number of cells in the grid
```

subSlice(self, *specs, **keys)
```
Get a subgrid based on an argument list <specs> of slices.
```

writeScrip(self, cdunifFile)
```
Write a grid to a SCRIP file
```

writeToFile(self, file)
```
Write self to a CdmsFile file, returning CF coordinates attri
```

Methods inherited from cdms.cdmsobj.CdmsObj:

dump(self, path=None, format=1)
```
dump(self,path=None,format=1)
Dump an XML representation of this object to a file.
'path' is the result file name, None for standard output.
'format'==1 iff the file is formatted with newlines for reada
```

matchPattern(self, pattern, attribute, tag)

```
    # Match a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must m
```

matchone(self, pattern, attname)

```
    # Return true iff the attribute with name attname is a string
    # attribute which matches the compiled regular expression pat
    # if attname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is
```

searchPattern(self, pattern, attribute, tag)

```
    # Search for a pattern in a string-valued attribute. If attri
    # search all string attributes. If tag is not None, it must m
```

searchPredicate(self, predicate, tag)

```
    # Apply a truth-valued predicate. Return a list containing a
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list
```

searchone(self, pattern, attname)

```
    Return true iff the attribute with name attname is a string
    attribute which contains the compiled regular expression patt
    if attname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is n
    a string.
```

Methods inherited from cdms.internattr.InternalAttributesClass:

is_internal_attribute(self, name)

```
    is_internal_attribute(name) is true if name is internal.
```

replace_external_attributes(self, newAttributes)

```
    replace_external_attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes
```

Methods inherited from PropertiedClasses.Properties.PropertiedClass:

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)

```
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.
```

get_property_g(self, name)

```
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.
```

get_property_s(self, name)

```
        Return the 'set' property handler for name that self uses.
        Returns None if no handler.
```

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)

```
        Set attribute handlers for  name to methods actg, acts, actd
        None means no change for that action.
        nowrite = 1 prevents setting this attribute.
            nowrite defaults to 0.
        nodelete = 1 prevents deleting this attribute.
            nodelete defaults to 1 unless actd given.
        if nowrite and nodelete is None: nodelete = 1
```

class ***DatasetCurveGrid***(AbstractCurveGrid)

Method resolution order:

DatasetCurveGrid
AbstractCurveGrid
AbstractHorizontalGrid
cdms.grid.AbstractGrid
cdms.cdmsobj.CdmsObj
cdms.internattr.InternalAttributesClass
PropertiedClasses.Properties.PropertiedClass

Methods defined here:

__init__(self, latAxis, lonAxis, id, parent=None, maskvar=None, tempmask=None, node=None)

```
        Create a file curvilinear grid.
```

__repr__(self)

Methods inherited from AbstractCurveGrid:

__str__ = __repr__(self)

checkAxes(self, axes)

```
        Return 1 iff every element of getAxisList() is in the list 'a
```

clone(self, copyData=1)

flatAxes(self)

```
        Return (flatlat, flatlon) where flatlat is a 1D NumPy array
        having the same length as the number of cells in the grid, si
        for flatlon.
```

genBounds(self)

```
        # Don't try to generate bounds for curvilinear grids
```

getAxis(self, naxis)

```
        # Get the n-th index axis. naxis is 0 or 1.
```

getAxisList(self)

getGridSlices(self, domainlist, newaxislist, slicelist)

```
Determine which slices in slicelist correspond to the lat/lon
of the grid.
domainlist is a list of axes of a variable.
newaxislist is a list of result axes after the slicelist is a
slicelist is a list of slices.

All lists are of equal length.

Return value is (newslicelist, gridaxislist) where
newslicelist is the elements of slicelist that correspond to
  preferred order of the grid.
gridaxislist is the elements of newaxislist that correspond t
  preferred order of the grid.
```

getIndex(self)

```
Get the grid index
```

getMask(self)

```
Get the mask array, if any, otherwise None is returned.
```

getMesh(self, transpose=None)

```
Generate a mesh array for the meshfill graphics method.
If transpose is defined to a tuple, say (1,0), first transpos
latbounds and lonbounds according to the tuple, (1,0,2) in th
```

intersect(self, spec)

```
Intersect with the region specification.

'spec' is a region specification of the form defined in the g

Returns (mask, indexspecs) where
'mask' is the mask of the result grid AFTER self and region s
'indexspecs' is a list of index specifications suitable for s
  variable with the given grid.
```

isClose(self, g)

```
Return 1 iff g is a grid of the same type and shape. A real e
comparison would be too expensive here.
```

reconcile(self, axes)

```
Return a grid that is consistent with the axes, or None.
For curvilinear grids this means that the grid-related axes a
contained in the 'axes' list.
```

size(self)

subSlice(self, *specs, **keys)

```
Get a transient subgrid based on an argument list <specs> of
```

toCurveGrid(self, gridid=None)

toGenericGrid(self, gridid=None)

writeScrip(self, cufile, gridTitle=None)

```
Write a grid to a SCRIP file.
cufile is a Cdunif file, NOT a CDMS file.
gridtitle is a string identifying the grid.
```

writeToFile(self, file)

Methods inherited from AbstractHorizontalGrid:

checkConvex(self)

```
Check that each cell of the grid is convex in lon-lat space,
Return a 1D Numeric array of cells that fail the cross-produc
```

fixCutCells(self, nonConvexCells, threshold=270.0)

```
For any mapping from a spherical to a planar surface, there i
Grid cells that span the cut may appear to be nonconvex, whic
problems with meshfill graphics. This routine attempts to 're
boundaries so that meshfill recognizes they are convex.

nonConvexCells: 1D Numeric array of indices of nonconvex cell
  checkConvex.
threshold: positive floating-point value in degrees.
  If the difference in longitude values of
  consecutive boundaries nodes exceeds the threshold, the cel
  a cut cell.

On return, the grid boundaries are modified.
Return value is a 1D array of indices of cells that cannot be
```

getBounds(self)

```
Get the grid cell boundaries, as a tuple (latitudeBounds, lon
```

getLatitude(self)

```
Get the latitude coordinates.
```

getLongitude(self)

```
Get the longitude coordinates.
```

getWeightsArray(self)

```
Return normalized area weights, as an array of the same
shape as the grid.
```

hasCoordType(self, coordType)

listall(self, all=None)

setMask(self, mask, permanent=0)

subGridRegion(self, latRegion, lonRegion)

Methods inherited from cdms.grid.AbstractGrid:

info(self, flag=None, device=None)

```
Write info about slab; include dimension values and weights i
```

Methods inherited from cdms.cdmsobj.CdmsObj:

dump(self, path=None, format=1)

```
dump(self,path=None,format=1)
Dump an XML representation of this object to a file.
'path' is the result file name, None for standard output.
'format'==1 iff the file is formatted with newlines for reada
```

matchPattern(self, pattern, attribute, tag)

```
# Match a pattern in a string-valued attribute. If attribute
# search all string attributes. If tag is not None, it must m
```

matchone(self, pattern, attname)

```
# Return true iff the attribute with name attname is a string
# attribute which matches the compiled regular expression pat
# if attname is None and pattern matches at least one string
# attribute. Return false if the attribute is not found or is
```

searchPattern(self, pattern, attribute, tag)

```
# Search for a pattern in a string-valued attribute. If attri
# search all string attributes. If tag is not None, it must m
```

searchPredicate(self, predicate, tag)

```
# Apply a truth-valued predicate. Return a list containing a
# if the predicate is true and either tag is None or matches
# If the predicate returns false, return an empty list
```

searchone(self, pattern, attname)

```
Return true iff the attribute with name attname is a string
attribute which contains the compiled regular expression patt
if attname is None and pattern matches at least one string
attribute. Return false if the attribute is not found or is n
a string.
```

Methods inherited from cdms.internattr.InternalAttributesClass:

is_internal_attribute(self, name)

```
is_internal_attribute(name) is true if name is internal.
```

replace_external_attributes(self, newAttributes)

```
replace_external_attributes(newAttributes)
Replace the external attributes with dictionary newAttributes
```

Methods inherited from PropertiedClasses.Properties.PropertiedClass:

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)
```
Return the 'del' property handler for name that self uses.
Returns None if no handler.
```

get_property_g(self, name)
```
Return the 'get' property handler for name that self uses.
Returns None if no handler.
```

get_property_s(self, name)
```
Return the 'set' property handler for name that self uses.
Returns None if no handler.
```

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
```
Set attribute handlers for  name to methods actg, acts, actd
None means no change for that action.
nowrite = 1 prevents setting this attribute.
    nowrite defaults to 0.
nodelete = 1 prevents deleting this attribute.
    nodelete defaults to 1 unless actd given.
if nowrite and nodelete is None: nodelete = 1
```

class ***FileCurveGrid***(AbstractCurveGrid)

Method resolution order:
- FileCurveGrid
- AbstractCurveGrid
- AbstractHorizontalGrid
- cdms.grid.AbstractGrid
- cdms.cdmsobj.CdmsObj
- cdms.internattr.InternalAttributesClass
- PropertiedClasses.Properties.PropertiedClass

Methods defined here:

__init__(self, latAxis, lonAxis, id, parent=None, maskvar=None, tempmask=None, node=None)
```
Create a file curvilinear grid.
```

__repr__(self)

Methods inherited from AbstractCurveGrid:

__str__ = __repr__(self)

checkAxes(self, axes)

```
        Return 1 iff every element of getAxisList() is in the list 'a
```

clone(self, copyData=1)

flatAxes(self)
```
        Return (flatlat, flatlon) where flatlat is a 1D NumPy array
        having the same length as the number of cells in the grid, si
        for flatlon.
```

genBounds(self)
```
        # Don't try to generate bounds for curvilinear grids
```

getAxis(self, naxis)
```
        # Get the n-th index axis. naxis is 0 or 1.
```

getAxisList(self)

getGridSlices(self, domainlist, newaxislist, slicelist)
```
        Determine which slices in slicelist correspond to the lat/lon
        of the grid.
        domainlist is a list of axes of a variable.
        newaxislist is a list of result axes after the slicelist is a
        slicelist is a list of slices.

        All lists are of equal length.

        Return value is (newslicelist, gridaxislist) where
        newslicelist is the elements of slicelist that correspond to
          preferred order of the grid.
        gridaxislist is the elements of newaxislist that correspond t
          preferred order of the grid.
```

getIndex(self)
```
        Get the grid index
```

getMask(self)
```
        Get the mask array, if any, otherwise None is returned.
```

getMesh(self, transpose=None)
```
        Generate a mesh array for the meshfill graphics method.
        If transpose is defined to a tuple, say (1,0), first transpos
        latbounds and lonbounds according to the tuple, (1,0,2) in th
```

intersect(self, spec)
```
        Intersect with the region specification.

        'spec' is a region specification of the form defined in the g

        Returns (mask, indexspecs) where
        'mask' is the mask of the result grid AFTER self and region s
        'indexspecs' is a list of index specifications suitable for s
          variable with the given grid.
```

isClose(self, g)

```
Return 1 iff g is a grid of the same type and shape. A real e
comparison would be too expensive here.
```

reconcile(self, axes)

```
Return a grid that is consistent with the axes, or None.
For curvilinear grids this means that the grid-related axes a
contained in the 'axes' list.
```

size(self)

subSlice(self, *specs, **keys)

```
Get a transient subgrid based on an argument list <specs> of
```

toCurveGrid(self, gridid=None)

toGenericGrid(self, gridid=None)

writeScrip(self, cufile, gridTitle=None)

```
Write a grid to a SCRIP file.
cufile is a Cdunif file, NOT a CDMS file.
gridtitle is a string identifying the grid.
```

writeToFile(self, file)

Methods inherited from AbstractHorizontalGrid:

checkConvex(self)

```
Check that each cell of the grid is convex in lon-lat space,
Return a 1D Numeric array of cells that fail the cross-produc
```

fixCutCells(self, nonConvexCells, threshold=270.0)

```
For any mapping from a spherical to a planar surface, there i
Grid cells that span the cut may appear to be nonconvex, whic
problems with meshfill graphics. This routine attempts to 're
boundaries so that meshfill recognizes they are convex.

nonConvexCells: 1D Numeric array of indices of nonconvex cell
  checkConvex.
threshold: positive floating-point value in degrees.
  If the difference in longitude values of
  consecutive boundaries nodes exceeds the threshold, the cel
  a cut cell.

On return, the grid boundaries are modified.
Return value is a 1D array of indices of cells that cannot be
```

getBounds(self)

```
Get the grid cell boundaries, as a tuple (latitudeBounds, lon
```

getLatitude(self)

```
Get the latitude coordinates.
```

getLongitude(self)
Get the longitude coordinates.

getWeightsArray(self)
Return normalized area weights, as an array of the same
shape as the grid.

hasCoordType(self, coordType)

listall(self, all=None)

setMask(self, mask, permanent=0)

subGridRegion(self, latRegion, lonRegion)

Methods inherited from cdms.grid.AbstractGrid:

info(self, flag=None, device=None)
Write info about slab; include dimension values and weights i

---

Methods inherited from cdms.cdmsobj.CdmsObj:

dump(self, path=None, format=1)
dump(self,path=None,format=1)
Dump an XML representation of this object to a file.
'path' is the result file name, None for standard output.
'format'==1 iff the file is formatted with newlines for reada

matchPattern(self, pattern, attribute, tag)
Match a pattern in a string-valued attribute. If attribute
search all string attributes. If tag is not None, it must m

matchone(self, pattern, attname)
Return true iff the attribute with name attname is a string
attribute which matches the compiled regular expression pat
if attname is None and pattern matches at least one string
attribute. Return false if the attribute is not found or is

searchPattern(self, pattern, attribute, tag)
Search for a pattern in a string-valued attribute. If attri
search all string attributes. If tag is not None, it must m

searchPredicate(self, predicate, tag)
Apply a truth-valued predicate. Return a list containing a
if the predicate is true and either tag is None or matches
If the predicate returns false, return an empty list

searchone(self, pattern, attname)
Return true iff the attribute with name attname is a string
attribute which contains the compiled regular expression patt
if attname is None and pattern matches at least one string

```
attribute. Return false if the attribute is not found or is n
a string.
```

Methods inherited from cdms.internattr.InternalAttributesClass:

is_internal_attribute(self, name)

```
is_internal_attribute(name) is true if name is internal.
```

replace_external_attributes(self, newAttributes)

```
replace_external_attributes(newAttributes)
Replace the external attributes with dictionary newAttributes
```

Methods inherited from PropertiedClasses.Properties.PropertiedClass:

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)

```
Return the 'del' property handler for name that self uses.
Returns None if no handler.
```

get_property_g(self, name)

```
Return the 'get' property handler for name that self uses.
Returns None if no handler.
```

get_property_s(self, name)

```
Return the 'set' property handler for name that self uses.
Returns None if no handler.
```

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)

```
Set attribute handlers for  name to methods actg, acts, actd
None means no change for that action.
nowrite = 1 prevents setting this attribute.
    nowrite defaults to 0.
nodelete = 1 prevents deleting this attribute.
    nodelete defaults to 1 unless actd given.
if nowrite and nodelete is None: nodelete = 1
```

class ***TransientCurveGrid***(AbstractCurveGrid)

Method resolution order:

TransientCurveGrid
AbstractCurveGrid
AbstractHorizontalGrid
cdms.grid.AbstractGrid
cdms.cdmsobj.CdmsObj
cdms.internattr.InternalAttributesClass

PropertiedClasses.Properties.PropertiedClass

Methods defined here:

__init__(self, latAxis, lonAxis, id=None, maskvar=None, tempmask=None)
```
Create a file curvilinear grid.
```

__repr__(self)

toCurveGrid(self, gridid=None)

Data and other attributes defined here:

grid_count = 0

Methods inherited from AbstractCurveGrid:

__str__ = __repr__(self)

checkAxes(self, axes)
```
Return 1 iff every element of getAxisList() is in the list 'a
```

clone(self, copyData=1)

flatAxes(self)
```
Return (flatlat, flatlon) where flatlat is a 1D NumPy array
having the same length as the number of cells in the grid, si
for flatlon.
```

genBounds(self)
```
# Don't try to generate bounds for curvilinear grids
```

getAxis(self, naxis)
```
# Get the n-th index axis. naxis is 0 or 1.
```

getAxisList(self)

getGridSlices(self, domainlist, newaxislist, slicelist)
```
Determine which slices in slicelist correspond to the lat/lon
of the grid.
domainlist is a list of axes of a variable.
newaxislist is a list of result axes after the slicelist is a
slicelist is a list of slices.

All lists are of equal length.

Return value is (newslicelist, gridaxislist) where
newslicelist is the elements of slicelist that correspond to
  preferred order of the grid.
gridaxislist is the elements of newaxislist that correspond t
  preferred order of the grid.
```

getIndex(self)
```
Get the grid index
```

getMask(self)
```
Get the mask array, if any, otherwise None is returned.
```

getMesh(self, transpose=None)
```
Generate a mesh array for the meshfill graphics method.
If transpose is defined to a tuple, say (1,0), first transpos
latbounds and lonbounds according to the tuple, (1,0,2) in th
```

intersect(self, spec)
```
Intersect with the region specification.

'spec' is a region specification of the form defined in the g

Returns (mask, indexspecs) where
'mask' is the mask of the result grid AFTER self and region s
'indexspecs' is a list of index specifications suitable for s
  variable with the given grid.
```

isClose(self, g)
```
Return 1 iff g is a grid of the same type and shape. A real e
comparison would be too expensive here.
```

reconcile(self, axes)
```
Return a grid that is consistent with the axes, or None.
For curvilinear grids this means that the grid-related axes a
contained in the 'axes' list.
```

size(self)

subSlice(self, *specs, **keys)
```
Get a transient subgrid based on an argument list <specs> of
```

toGenericGrid(self, gridid=None)

writeScrip(self, cufile, gridTitle=None)
```
Write a grid to a SCRIP file.
cufile is a Cdunif file, NOT a CDMS file.
gridtitle is a string identifying the grid.
```

writeToFile(self, file)

Methods inherited from AbstractHorizontalGrid:

checkConvex(self)
```
Check that each cell of the grid is convex in lon-lat space,
Return a 1D Numeric array of cells that fail the cross-produc
```

fixCutCells(self, nonConvexCells, threshold=270.0)

```
    For any mapping from a spherical to a planar surface, there i
    Grid cells that span the cut may appear to be nonconvex, whic
    problems with meshfill graphics. This routine attempts to 're
    boundaries so that meshfill recognizes they are convex.

    nonConvexCells: 1D Numeric array of indices of nonconvex cell
      checkConvex.
    threshold: positive floating-point value in degrees.
      If the difference in longitude values of
      consecutive boundaries nodes exceeds the threshold, the cel
      a cut cell.

    On return, the grid boundaries are modified.
    Return value is a 1D array of indices of cells that cannot be
```

getBounds(self)

```
    Get the grid cell boundaries, as a tuple (latitudeBounds, lon
```

getLatitude(self)

```
    Get the latitude coordinates.
```

getLongitude(self)

```
    Get the longitude coordinates.
```

getWeightsArray(self)

```
    Return normalized area weights, as an array of the same
    shape as the grid.
```

hasCoordType(self, coordType)

listall(self, all=None)

setMask(self, mask, permanent=0)

subGridRegion(self, latRegion, lonRegion)

Methods inherited from cdms.grid.AbstractGrid:

info(self, flag=None, device=None)

```
    Write info about slab; include dimension values and weights i
```

Methods inherited from cdms.cdmsobj.CdmsObj:

dump(self, path=None, format=1)

```
    dump(self,path=None,format=1)
    Dump an XML representation of this object to a file.
    'path' is the result file name, None for standard output.
    'format'==1 iff the file is formatted with newlines for reada
```

matchPattern(self, pattern, attribute, tag)

```
    # Match a pattern in a string-valued attribute. If attribute
```

```
    # search all string attributes. If tag is not None, it must m
```

matchone(self, pattern, attname)

```
    # Return true iff the attribute with name attname is a string
    # attribute which matches the compiled regular expression pat
    # if attname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is
```

searchPattern(self, pattern, attribute, tag)

```
    # Search for a pattern in a string-valued attribute. If attri
    # search all string attributes. If tag is not None, it must m
```

searchPredicate(self, predicate, tag)

```
    # Apply a truth-valued predicate. Return a list containing a
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list
```

searchone(self, pattern, attname)

```
    Return true iff the attribute with name attname is a string
    attribute which contains the compiled regular expression patt
    if attname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is n
    a string.
```

Methods inherited from cdms.internattr.InternalAttributesClass:

is_internal_attribute(self, name)

```
    is_internal_attribute(name) is true if name is internal.
```

replace_external_attributes(self, newAttributes)

```
    replace_external_attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes
```

Methods inherited from PropertiedClasses.Properties.PropertiedClass:

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)

```
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.
```

get_property_g(self, name)

```
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.
```

get_property_s(self, name)

```
    Return the 'set' property handler for name that self uses.
```

```
        Returns None if no handler.
```

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)

```
        Set attribute handlers for  name to methods actg, acts, actd
        None means no change for that action.
        nowrite = 1 prevents setting this attribute.
            nowrite defaults to 0.
        nodelete = 1 prevents deleting this attribute.
            nodelete defaults to 1 unless actd given.
        if nowrite and nodelete is None: nodelete = 1
```

Functions

readScripCurveGrid(fileobj, dims, whichType, whichGrid)

```
        Read a 'native' SCRIP grid file, returning a transient curvilinear
        fileobj is an open CDMS dataset or file object.
        dims is the grid shape.
        whichType is the type of file, either "grid" or "mapping"
        if whichType is "mapping", whichGrid is the choice of grid, either
```

Data

CoordTypeToLoc = {'lat': 1, 'lev': 2, 'lon': 0}
LatitudeType = 'lat'
LongitudeType = 'lon'
MethodNotImplemented = 'Method not yet implemented'
TimeType = 'time'
VerticalType = 'lev'